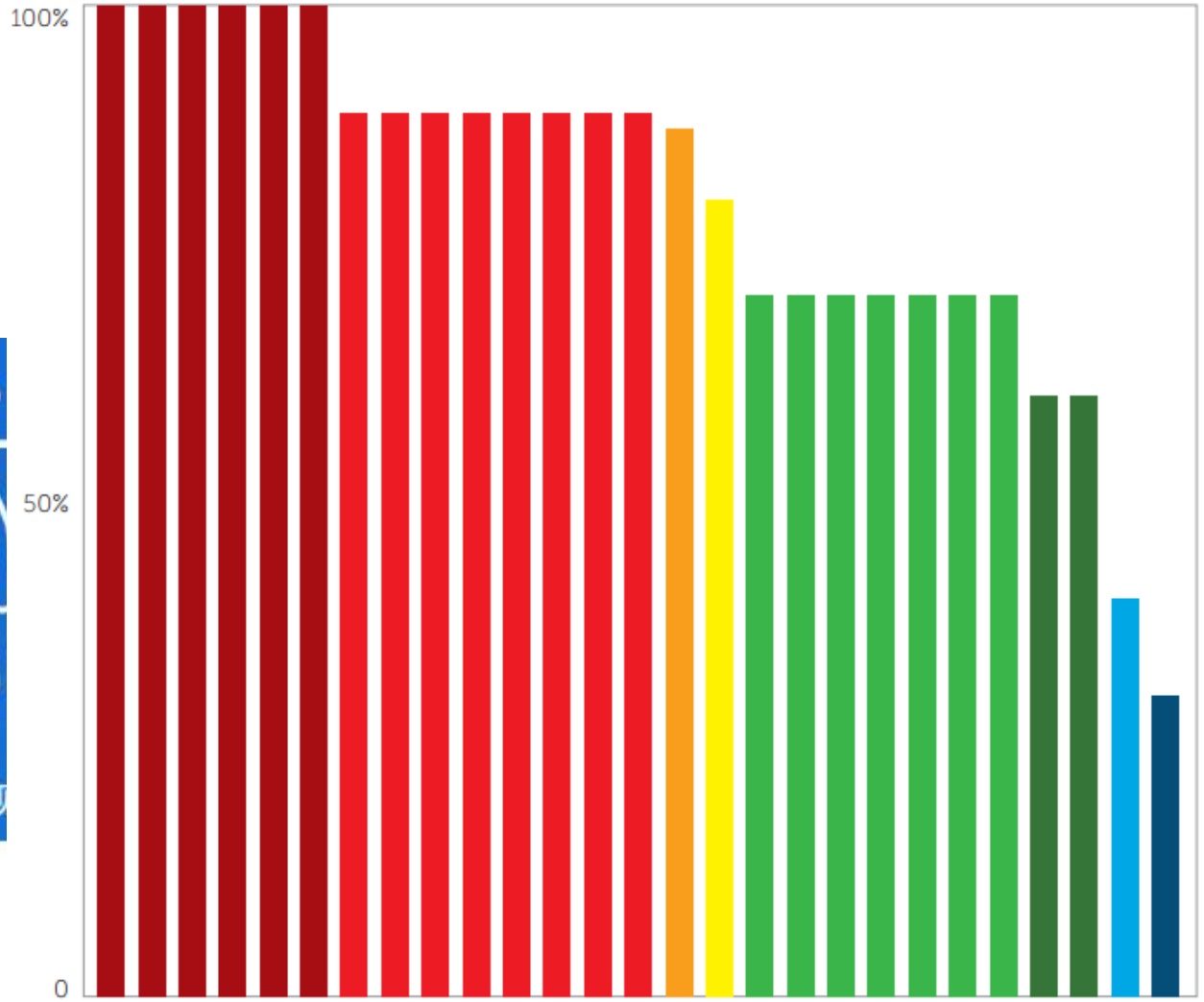
The Terminator to Android Hardening Services

Yueqian Zhang, Xiapu Luo, Haoyang Yin
Department of Computing
The Hong Kong Polytechnic University



Widgets	100%
Media & Video	100%
Lifestyle	100%
Finance	100%
Travel & Local	100%
Tools	100%
Shopping	90%
Business	90%
Music & Audio	90%
Photography	90%
Weather	90%
Transportation	90%
Productivity	90%
News & Magazines	90%
Health & Fitness	89%
Personalization	80%
Entertainment	70%
Games	70%
Books & Reference	70%
Comics	70%
Communication	70%
Live Wallpaper	70%
Social	70%
Sports	60%
Education	60%
Medical	40%
Libraries & Demo	30%

urce:ijiami

Source: Trend Micro

Percentage of top 10 apps in each category which have repacked version:

- **100% of the apps of Widgets, Media & Video, etc.**
- **90% of the apps of Business, Music & Audio, etc.**
- **…**


RSACONFE
FEBRUARY 24 – 28 | MOSC
Beginner
Android A
SESSION ID: STU-W02B
Share.
Learn.
Secure.
Capitalizing on
Collective Intelligence
Android及
ARM原生語言 [第二版]
逆向工程破解Android APP安全
別讓你的程式碼成為別人的砲灰
暨生強 編著
本書範例，
可至佳魁資訊官網下載
佳魁資訊
just minutes to reverse engineer Android apps.


防篡改
防二次打包
防破解
防伪造
反调试
防反编译
360加固保 未雨绸缪保安全
—防止APP被恶意篡改
—防止APP被植入恶意代码、广告
—防止APP被反编译、二次打包
—时刻保护开发者和用户的利益
马上加固


一站式解决您应用的安全问题
特有的应用安全全面防护方案，可以从内到外完美解决应用安全加密，安全存储，安全加签，反调试，反篡改等难题


du
应用加固
反静态分析，反动态调试，
反内存窃取，反恶意篡改...


梆梆 for Android 保护你的App
• 防止App被篡改
• 防止App反编译
• 防止App被动态注入
• 防止App数据被窃取


APKTOOL
gdb
IDA Pro
1 有人要破解我 好害怕
2 爱加密移动应用安全保护平台
3 再也不担心安全问题啦


APP
抗盗版神器
腾讯云应用加固
渠道监控 · 加固保护 · 漏洞检测
立即体验

Outline

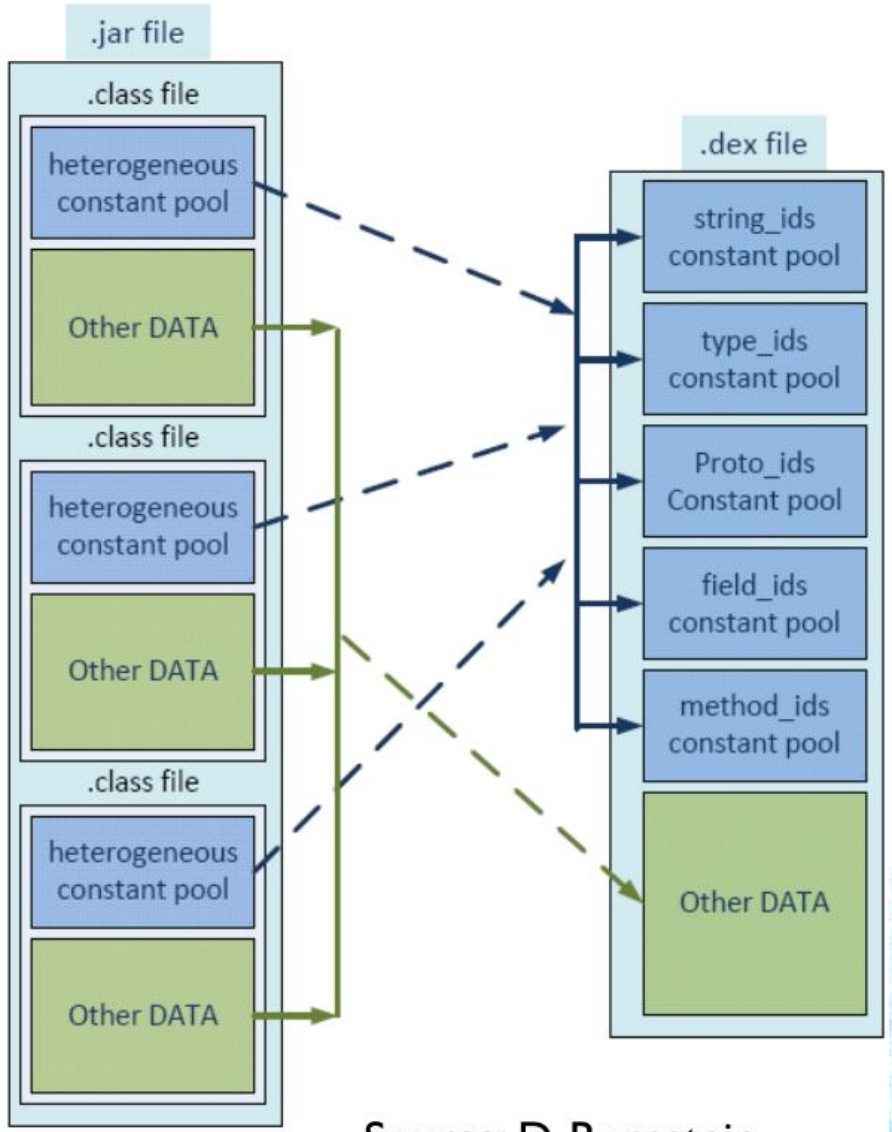
- Background
- DexHunter
- Analysis of major products
- Related work

Outline

- Background
- DexHunter
- Analysis of major products
- Related work

Dex File

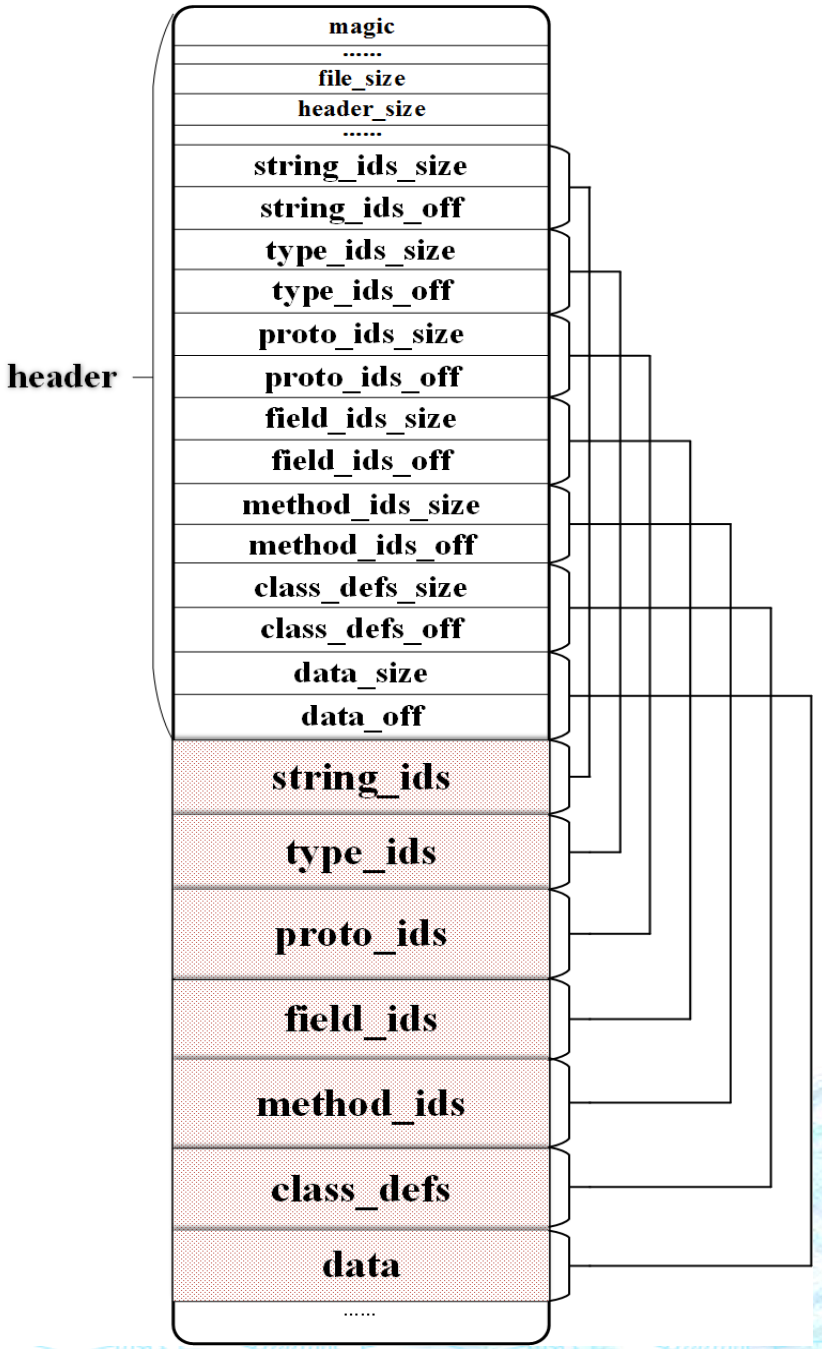
- Java source code -> Java class -> dex
  - Java class: each file contains one class
  - dex: one file contains all classes
- Reorganize constant pools in each class file into shared and type-specific constant pools



Source: D. Bornstein

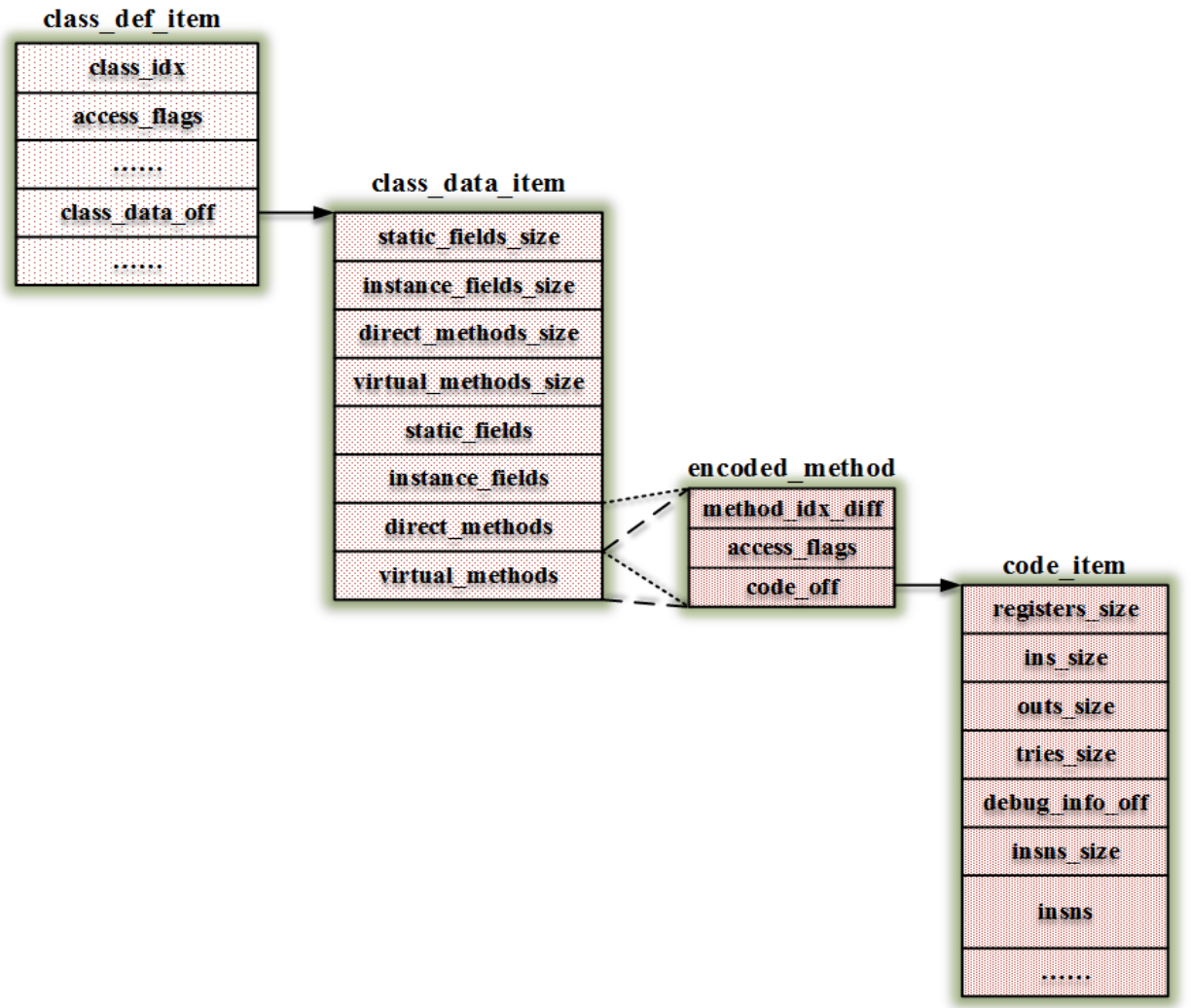
Dex File

- The executable of an App.
- The header contains the length and the offset for each section.
- ***class_defs*** section contains ***class_def_item***s, each of which describes a class.



class_def_item

- A ***class_def_item*** points to a ***class_data_item***.
- A ***class_data_item*** contains the data of a class.
- Each method is described by an ***encoded_method***.
- An ***encoded_method*** points to a ***code_item***.
- A ***code_item*** contains the instructions of a method.



OAT File

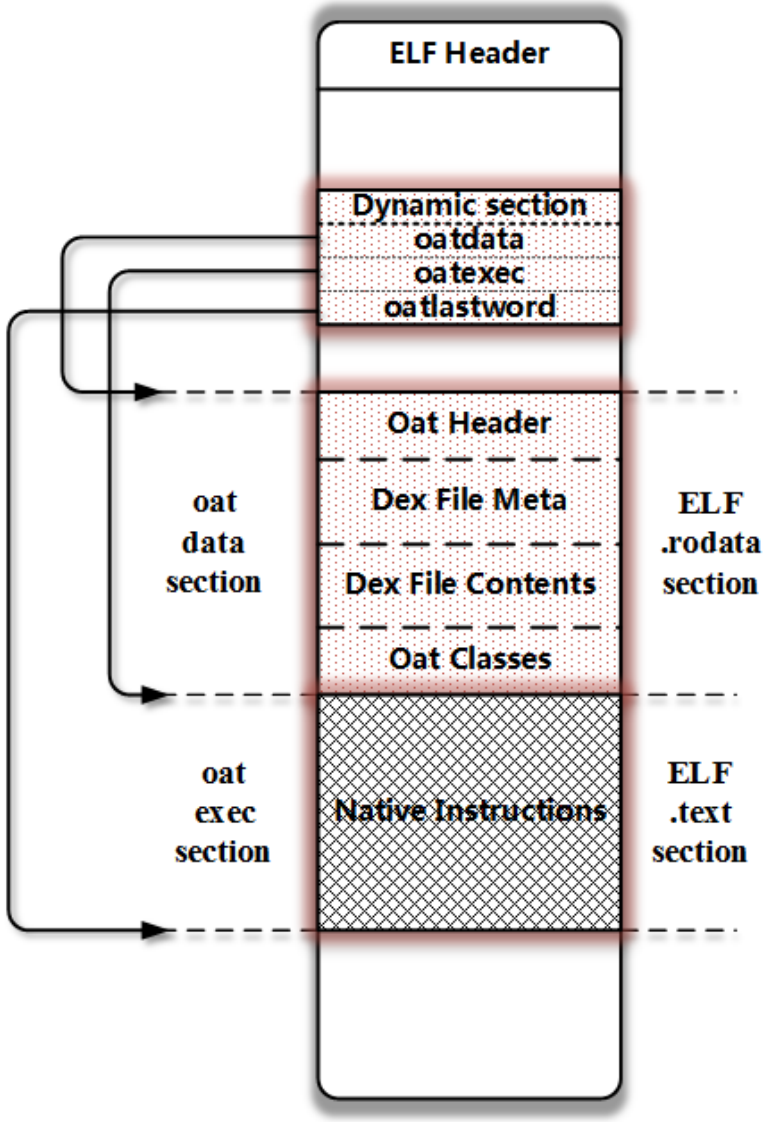
- It is generated while an app is installed or a jar file is loaded.
  - /frameworks/base/services/java/com/android/server/pm/PackageManagerService.java
  - Constructor method → ***scanDirLI ()→ scanPackageLI()→performDexOptLI()→mInstaller.dexopt()***

- It is an ELF file.

```
system@priv-app@VoiceDialer.apk@classes.dex: ELF 32-bit LSB  shared object, ARM,
 EABI5 version 1 (GNU/Linux), dynamically linked, stripped
```

OAT File

- Three symbols in dynamic section.
  - oatdata
  - oatexec
  - oatlastword
- The original dex file is contained in the ***oatdata*** section.
- The compiled native instructions are contained the ***oatexec*** section.



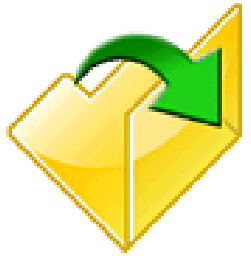
Outline

- Background
- DexHunter
  - Where to unpack the app?
  - When to unpack the app?
  - How to unpack the app?
- Analysis of major products
- Related work

Where to dump dex file?

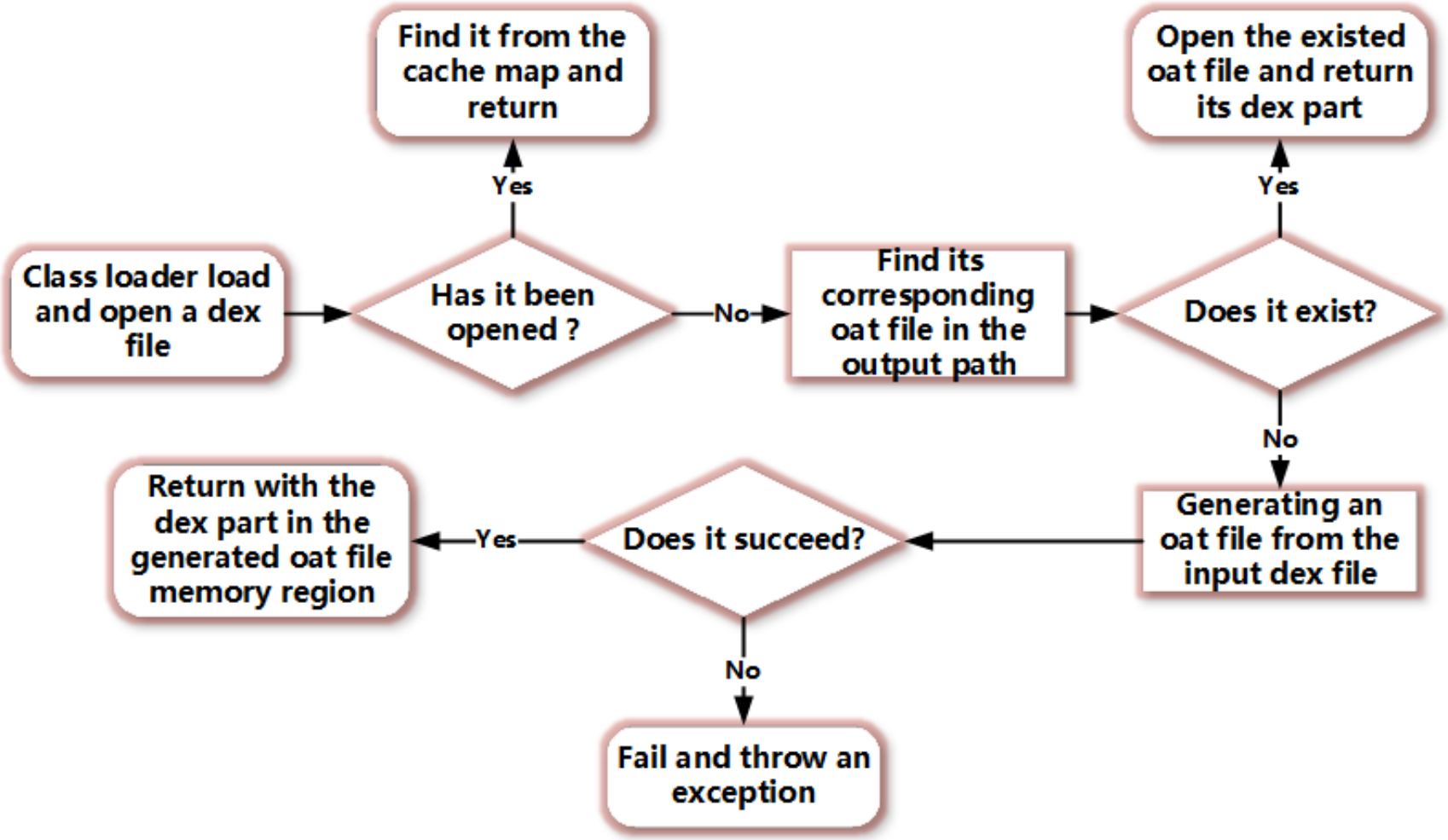
- Four occasions
  - Opening a Dex file;
  - Loading a class;
  - Initializing a class;
  - Invoking a method;

Opening a Dex File

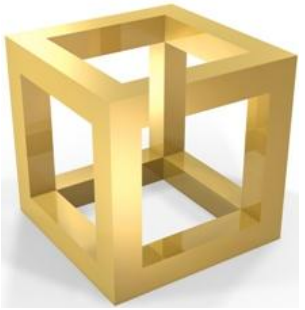


- Operations
  - Open an APK file;
  - Check whether it has been cached;
  - If not, extract the dex file from the APK and generate the cached dex file;
  - Open the cached dex file.

Procedure of Opening a Dex File in ART



Loading a Class



- Operations
  - Form a class object from the data;
  - Verify the legitimacy of access flags and the data;
  - Populate all fields in the class object;
  - Deal with its super classes and/or interfaces;
  - Conduct some other checking.

Two Ways of Loading a Classes

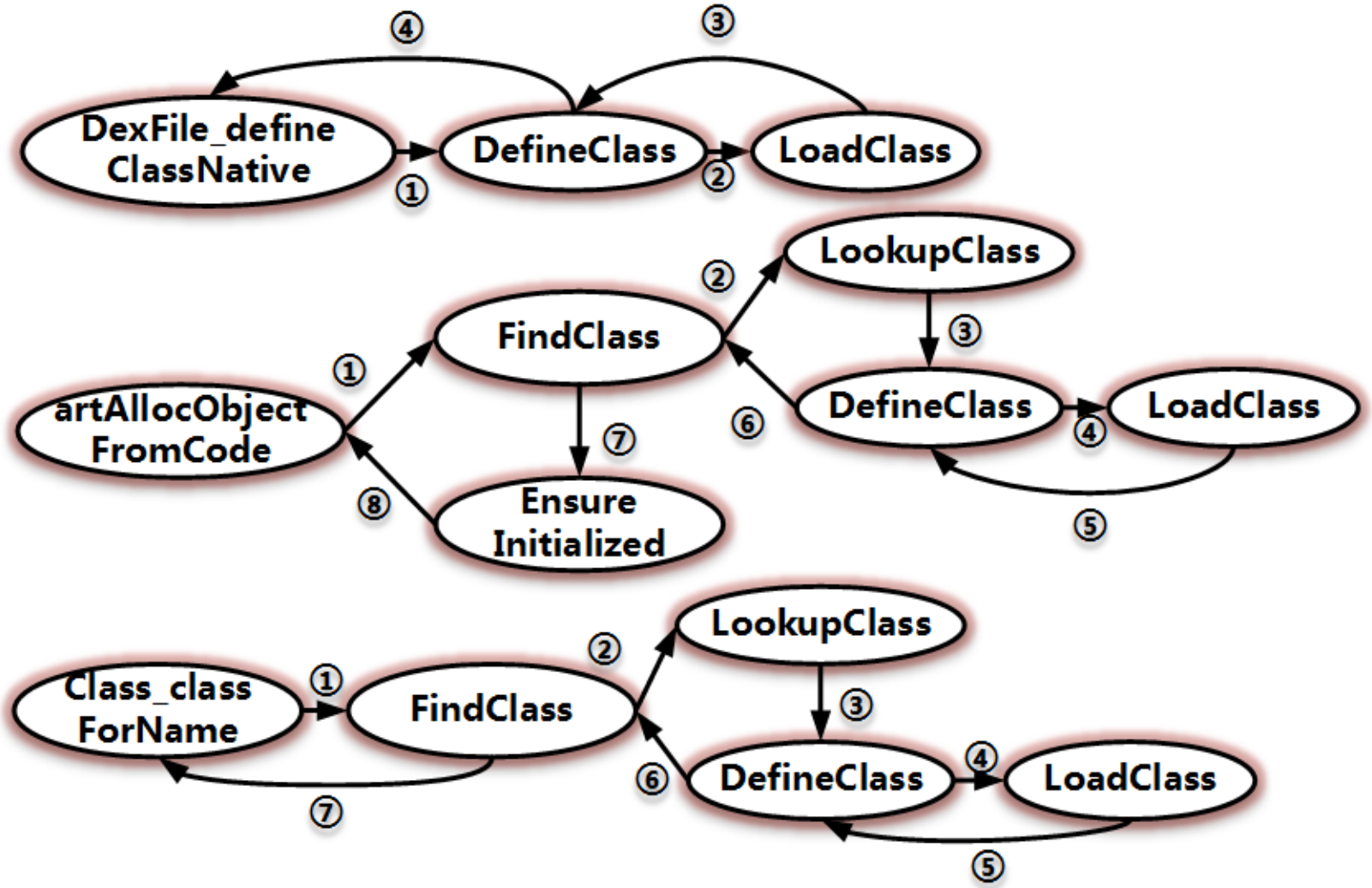
- Explicit approach
  - ***Class.forName()***, ***ClassLoader.loadClass()***.

- Implicit approach
  - E.g., ***new*** operation, accessing static members, etc.

Implementation in ART

- Explicit
  - ***ClassLoader.loadClass →DexFile_defineClassNative***
  - ***Class.forName →Class_classForName***
- Implicit
  - ***new*** operations and so on→ ***artAllocObjectFromCode***

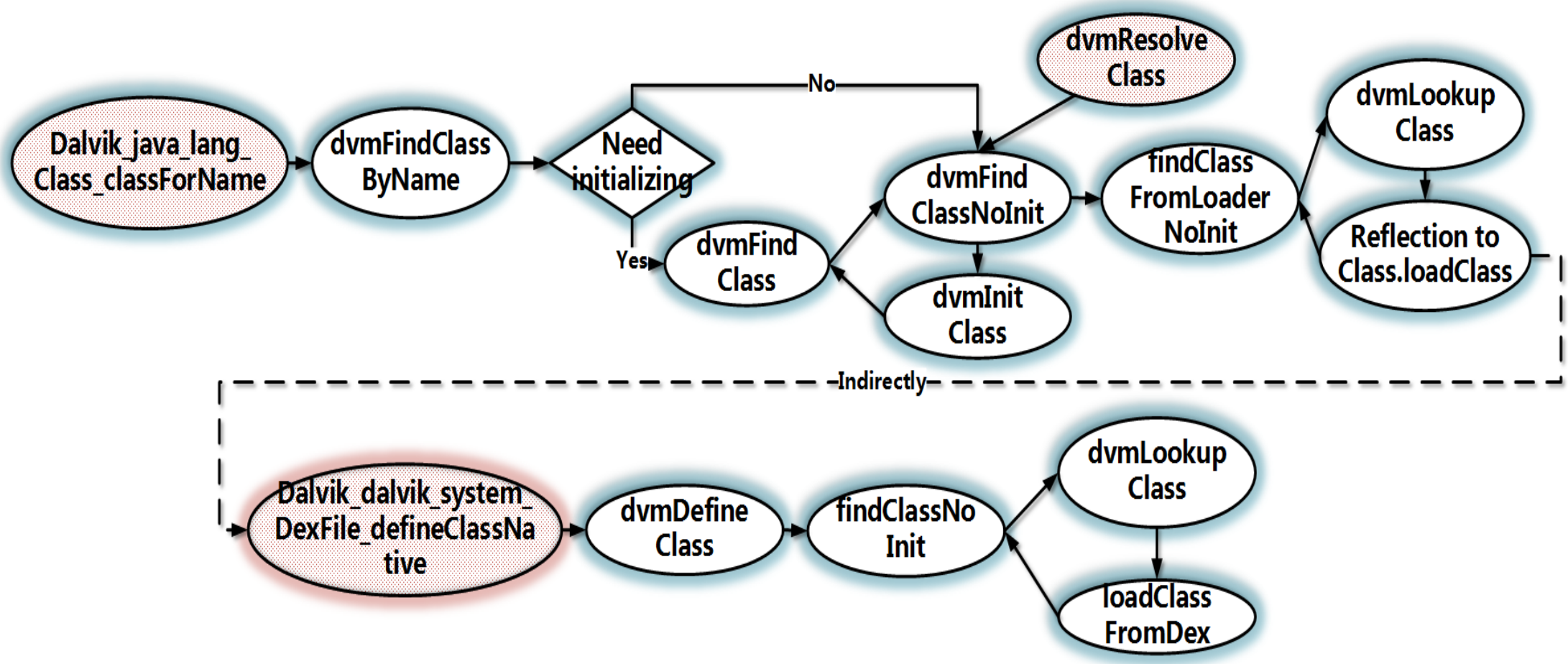
Implementation in ART



Implementation in DVM

- Explicit
  - ***ClassLoader.loadClass→Dalvik_dalvik_system_DexFile defineClassNative***
  - ***Class.forName →Dalvik_java_lang_Class_classForName***
- Implicit
  - ***new*** operations and so on→ ***dvmResolveClass***

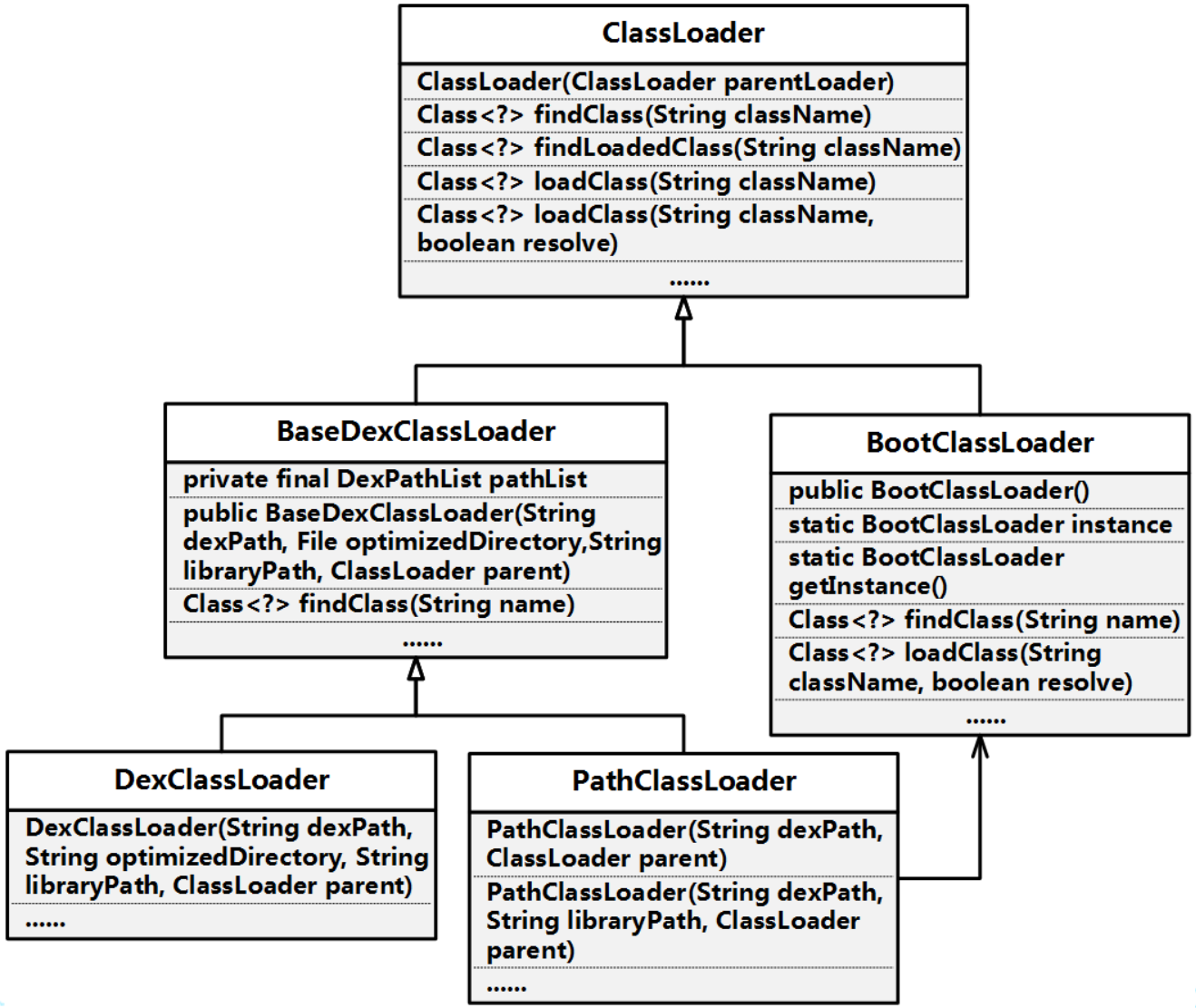
Implementation in DVM



Class Loaders at Java Level

- Three class loaders
  - ***BootClassLoader***
    - It is used for loading system classes.
  - ***DexClassLoader***
    - It is used for loading external files.
  - ***PathClassLoader***
    - It is used by the framework.

Inheritance Relationship



Parent Delegation Model

```
Class<?> loadClass(String className, boolean resolve {
    Class<?> clazz = findLoadedClass(className);
    if (clazz == null) {
        clazz = parent.loadClass(className, false);
    }
    if (clazz == null) {
        clazz = findClass(className);
    }
    }
    return clazz;
}
```

Parent Delegation Model

- Each subclass of ClassLoader implements its own ***findClass***().

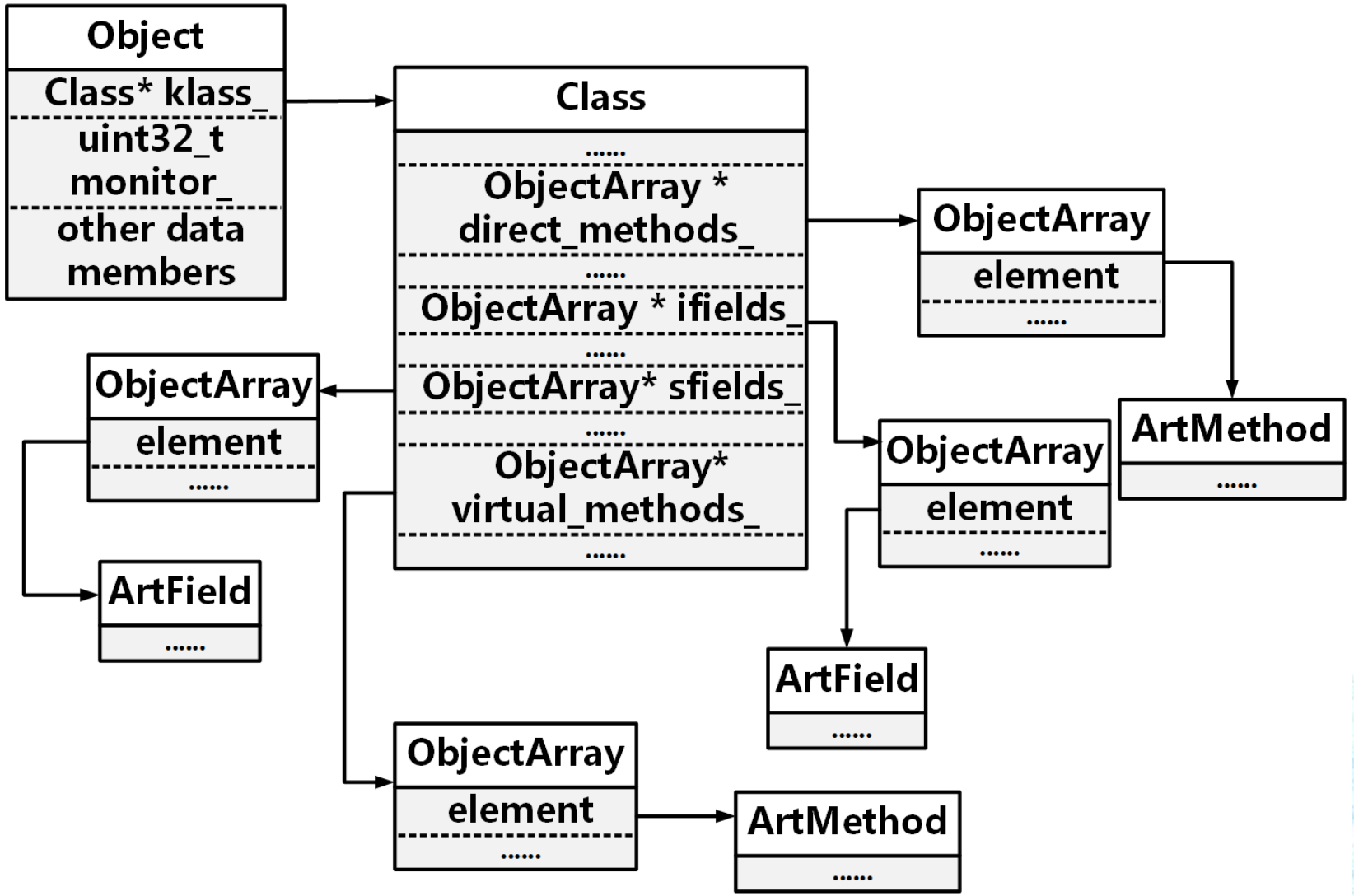
- Each subclass of ClassLoader inherits ***loadClass***() except ***BootClassLoader***.

Differences between Java and Android

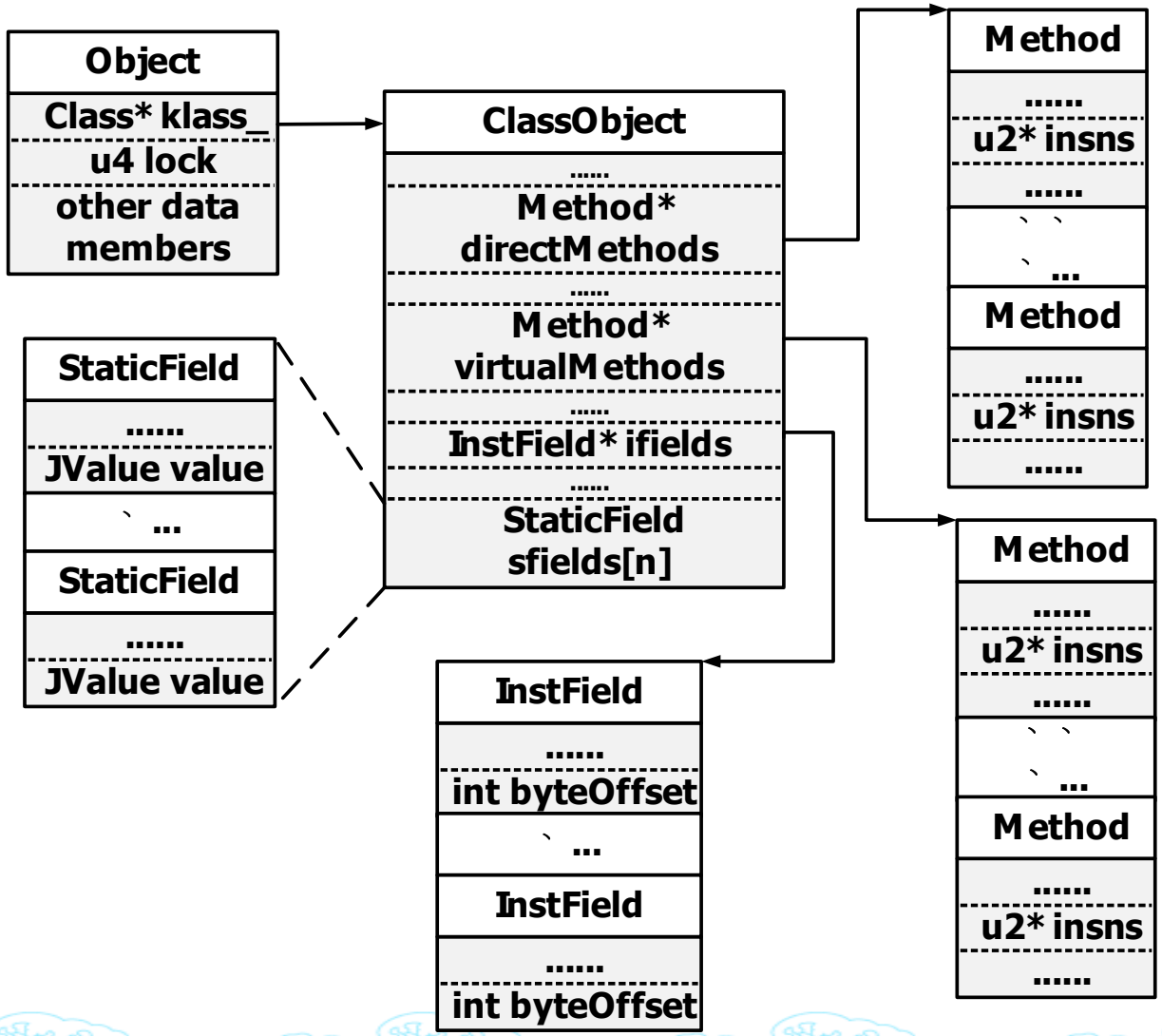
- ***defineClass()*** in ***ClassLoader*** (Android) is not implemented.
  - Throw **UnsupportedOperationException**

- ***URLClassLoader*** in Android also cannot load a class, because
  - ***URLClassLoader.findClass()*** →
    URLHandler/URLJarHandler.findClass()→
    createClass ()→
    ClassLoader.defineclass()

A Loaded Class Object in ART



A Loaded Class Object in DVM



When does Initializing Classes happen?

- Before the class object is used;
- Before the first static data member is accessed;
- Before the first static method is invoked;
- Before the first instance is generated;
- …

Invoking a Method

- DVM or ART interpreting mode
  - Execute the instructions in the ***code_item***.

- ART native mode
  - Execute the native instructions in ***oatexec*** section.

When to unpack the app?

- When the first class of the app is being loaded.

- Why?
  - Before a class is loaded, the content of the class should be available in the memory;

  - When the class is initialized, some content in memory may be modified dynamically;

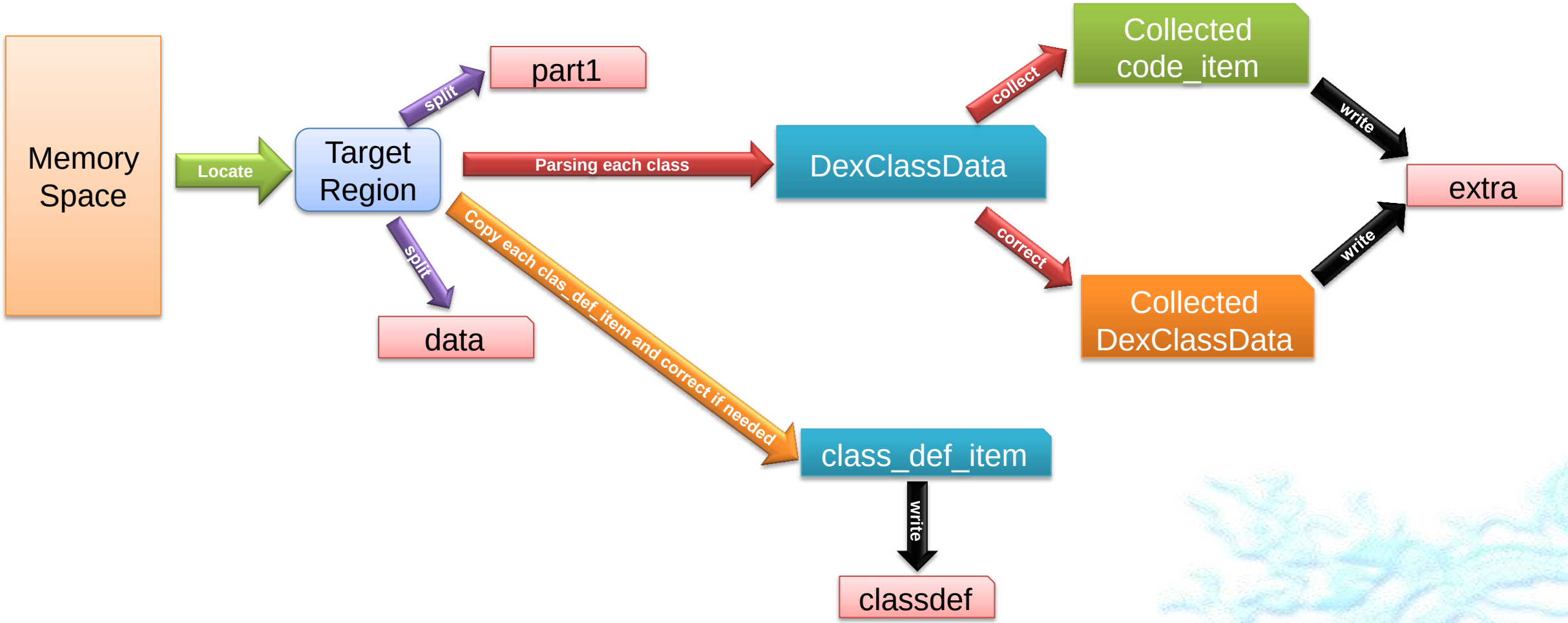
  - Just before a method is invoked, its ***code_item*** or instructions should be available.

- How?
  - Load and initialize all classes proactively.

How to unpack the apk?

- Integrate our tool into Android runtime including DVM and ART.
- Wait for the proper occasion.
- Locate the target memory region.
- Dump the selected memory.
- Correct and reconstruct the dex file.

DexHunter


Memory Space
Locate
Target Region
split
part1
split
data
Parsing each class
DexClassData
collect
Collected code_item
correct
Collected DexClassData
write
write
extra
Copy each clas_def_item and correct if needed
class_def_item
write
classdef

Loading & Initializing Classes

- Traverse all ***class_def_item***s in the dex file.
- For each one, we load it with ***FindClass*** function (ART) or ***dvmDefineClass*** function (DVM).
- Then we initialize it with ***EnsureInitialized*** function (ART) or ***dvmIsClassInitialized*** & ***dvmInitClass*** functions (DVM).

Locating the Target Memory Region

- The target memory region contains the dex file.

- We use a special string to determine whether the current dex file is what we want.

The Special String in ART

- ART: the string "**location_**" in ***DexFile*** objects.

- The opened apk file's path →
  dex_file_location in generated oat file's header
  → ***dex_file_location_*** in ***OatDexFile*** objects
  → ***location_*** in DexFile objects by function ***DexFile::Open***

The Special String in DVM

- DVM: the string "**fileName**" in ***DexOrJar*** objects.
- The opened apk file path→
  fileName in ***DexOrJar*** objects by function ***Dalvik_dalvik_system_DexFile_openDexFileNative***.
- For ***Dalvik_dalvik_system_DexFile_openDexFile_bytearray***, fileName is always equal to "***<memory>***".

Extracting the Dex File in Memory

- Divide the target memory region
  - Part 1: the content before the ***class_defs*** section
  - Part 2: the ***class_defs*** section
  - Part 3: the content after the ***class_defs*** section
- Dump part 1 into a file named **part1** and part 3 into a file named **data**.

Parsing the Content

- Parse ***class_defs*** section.
- Getting each ***class_data_item*** from ***class_def_item***.
- Read the corresponding content into a ***DexClassData*** object.
- Notice: some fields in a ***class_data_item*** are encoded by LEB128 algorithm.

```
struct DexClassData {  // For one class_def_item
    DexClassDataHeader header;
    DexField*     staticFields;
    DexField*     instanceFields;
    DexMethod*    directMethods;
    DexMethod*    virtualMethods;
};

struct DexField {  //For one field
    uint32_t delta_fieldIdx;
    uint32_t accessFlags;
};
```

```
struct DexClassDataHeader { // For one header
    uint32_t staticFieldsSize;
    uint32_t instanceFieldsSize;
    uint32_t directMethodsSize;
    uint32_t virtualMethodsSize;
};

struct DexMethod {  //For one method
    uint32_t delta_methodIdx;
    uint32_t accessFlags;
    uint32_t codeOff;
};
```

Correcting and Collecting

- Why?
  - Packing services may modify the memory dynamically.
  - The memory consists of the region containing the dex file and the method objects (i.e., ***ArtMethod*** in ART, ***Method*** in DVM) managed by runtime.
  - The runtime executes instructions according to the managed method objects.

Correcting and Collecting

- We check each:
  - ***class_data_off*** in ***class_def_item***.
  - ***accessflag*** and ***codeoff*** in ***DexMethod*** of parsed ***class_data_item*** (i.e., ***DexClassData*** object).

How?

- Determine whether the ***class_data_off*** in ***class_def_item*** exists in the scope of the dex file.
  - Copy all ***class_def_item***s and write them into a file named **classdef**.
  - Collect the outside ***class_data_item***s into a file named **extra**.

- Correct the fields in selected ***DexClassData*** object according to the managed method object.

Scenario I

- Compare the ***accessFlags*** in ***DexMethod*** with the access flag in the managed method object.
- Compare the ***codeoff*** in ***DexMethod*** with the ***code_item_off*** in the managed method object.
- If at least one is not equal, we modify the value in the ***DexMethod*** object according to the managed method object and write the relevant ***DexClassData*** into **extra** file.

Scenario II

- Check whether ***code_item_off*** exists in the scope of the dex file.

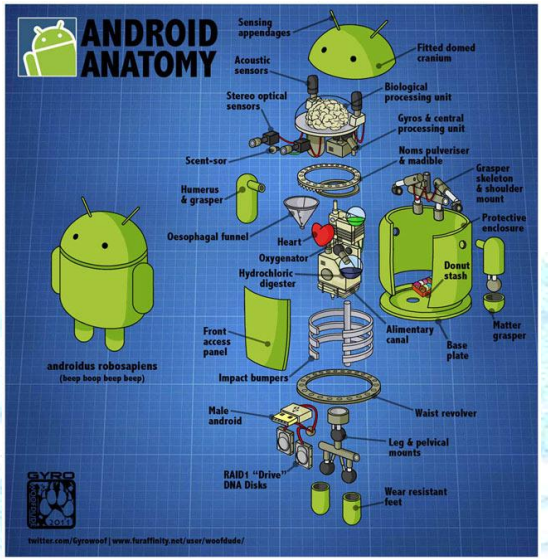
- If not, we collect the correct ***code_item*** and write it into **extra** file.

Reconstructing the Dex File

- We now have four files: **part1**, **classdef**, **data**, **extra**.

- We combine them as the sequence
  (1) **part1**
  (2) **classdef**
  (3) **data**
  (4) **extra**

- Finally, we obtain a complete dex file.



Outline

- Background
- DexHunter
- Analysis of major products
- Related work

Products under Investigation

- 360 http://jiagu.360.cn/
- Ali http://jaq.alibaba.com/
- Baidu http://apkprotect.baidu.com/
- Bangcle http://www.bangcle.com/
- Tencent http://jiagu.qcloud.com/
- ijiami http://www.ijiami.cn/

Experiment Setup

nexus 4

String List

360	**/data/data/XXX/.jiagu/classes.dex**
Ali	**/data/data/XXX/files/libmobisecy1.zip**
Baidu	**/data/data/XXX/.1/classes.jar**
Bangcle	**/data/data/XXX/.cache/classes.jar**
Tencent	**/data/app/XXX-1.apk (/data/app/XXX-2.apk)**
ijiami	**/data/data/XXX/cache/.**

XXX stands for its package name.

Anti-debugging

- All products detect debugger
- Anti-ptrace
- Anti-JWDP
- ....
- They cannot detect DexHunter.

360

360加固保
jiagu.360.cn

- Version: 06-21-2015
- It encrypts the dex file and saves it in libjiagu.so/libjiagu_art.so.
- It releases the data into memory and decrypts it while running.

- Version: 21-06-2015
- It splits the original dex file into two parts
  - One is the main body saved in libmobisecy.so
  - The other one contains the ***class_data_item***s and the ***code_item***s of some ***class_def_item***s.
- It releases both two parts into memory as plain text and corrects some offset values in the main body while running.
- Some ***annotation_off***s are set to incorrect values.

Baidu

- Version: 21-06-2015
- It moves some class_data_items to other places outside the dex file.
- It wipes the magic numbers, checksum and signature in the header after the dex file has been opened.

Baidu

- It fills in an empty method just before it is invoked and erases the content after the method is finished.

- We instrument method invocation to dump these methods which is available only just before invoking.
  - ***DoInvoke*** (ART)
  - ***dvmMterp_invokeMethod*** (DVM)

Bangcle

- Version: 21-06-2015
- It prepares the odex file or oat file in advance.
- It encrypts the file and stores it in an external jar file.
- It decrypts the data while running
- It hooks several functions in libc.so, such as
  - fwrite, mmap, …

ijiami

爱加密
IJIAMI.CN

- Version: 21-06-2015
- Similar to Bangcle
- The string changes every time the app runs.
- It releases the decrypted file, which is also encrypted as a jar file, with different file names each time while they are in the same directory.

Tencent

腾讯云 应用加固

- Version: 25-05-2015
- It can protect the methods selected by users.
- If a method is selected, it cannot be found in the relevant ***class_data_item***.
- It releases the real ***class_data_item*** and adjusts the offset.
  - The ***code_item*** of the selected method is still in the **data** section.
- Some ***annotation_off***s and ***debug_info_off***s are set to 0xFFFFFFFF.
- It can only runs in DVM.

Outline

- Background
- DexHunter
- Analysis of Major Products
- Related work

Related work

- A. Apvrille and R. Nigam, "Obfuscation in android malware, and how to fight back," Virus Bulletin, July 2014.
- M. Grassi, "Reverse engineering, pentesting, and hardening of android apps." DroidCon, 2014.
- T. Strazzere and J. Sawyer, "Android hacker protection level 0," DefCon, 2014. (android-unpacker, https://github.com/strazzere/android-unpacker)
- ZjDroid, http://blog.csdn.net/androidsecurity/article/details/38121585
- Y. Park, "We can still crack you! general unpacking method for android packer (no root)," Blackhat Asia, 2015.
- Y. Shao et al., DexDumper in paper "Towards a Scalable Resource-driven Approach for Detecting Repackaged Android Applications", Proc. ACSAC, 2014.

DEMO


thank
you!